-- --

IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

4.50 `printf` — Write formatted output

4.50.1 Synopsis

`printf` *format* [*argument* ...]

4.50.2 Description

The `printf` utility shall write formatted operands to the standard output. The *argument* operands shall be formatted under control of the *format* operand.

4.50.3 Options

None.

4.50.4 Operands

The following operands shall be supported by the implementation:

format — A string describing the format to use to write the remaining operands; see 4.50.7.

argument — The strings to be written to standard output, under the control of *format* ; see 4.50.7.

4.50.5 External Influences

4.50.5.1 Standard Input

None.

4.50.5.2 Input Files

None.

4.50.5.3 Environment Variables

The following environment variables shall affect the execution of `printf`:

LANG — This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

LC_NUMERIC This variable shall determine the locale for numeric formatting. It shall affect the format of numbers written using the `e`, `E`, `f`, `g`, and `G` conversion characters (if supported).

4.50.5.4 Asynchronous Events

Default.

4.50.6 External Effects

4.50.6.1 Standard Output

See 4.50.7.

4.50.6.2 Standard Error

Used only for diagnostic messages.

4.50.6.3 Output Files

None.

4.50.7 Extended Description

The *format* operand shall be used as the *format* string described in 2.12 with the following exceptions:

(1) A <space> character in the format string, in any context other than a flag of a conversion specification, shall be treated as an ordinary character that is copied to the output.

(2) A Δ character in the format string shall be treated as a Δ character, not as a <space>.

(3) In addition to the escape sequences shown in Table 2-15 (see 2.12), *ddd*, where *ddd* is a one-, two-, or three-digit octal number, shall be written as a byte with the numeric value specified by the octal number.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(4) The implementation shall not precede or follow output from the `d` or `u` conversion specifications with `<blank>`s not specified by the *format* operand.

(5) The implementation shall not precede output from the `o` conversion specification with zeroes not specified by the *format* operand.

(6) The `e`, `E`, `f`, `g`, and `G` conversion specifications need not be supported.

(7) An additional conversion character, `b`, shall be supported as follows. The argument shall be taken to be a string that may contain backslash-escape sequences. The following backslash-escape sequences shall be supported:

(a) The escape sequences listed in Table 2-15, which shall be converted to the characters they represent;

(b) `\0`*ddd*, where *ddd* is a zero-, one-, two-, or three-digit octal number that shall be converted to a byte with the numeric value specified by the octal number;

(c) `\c`, which shall not be written and shall cause `printf` to ignore any remaining characters in the string operand containing it, any remaining string operands, and any additional characters in the *format* operand.

The interpretation of a backslash followed by any other sequence of characters is unspecified.

Bytes from the converted string shall be written until the end of the string or the number of bytes indicated by the precision specification is reached. If the precision is omitted, it shall be taken to be infinite, so all bytes up to the end of the converted string shall be written.

(8) For each specification that consumes an argument, the next argument operand shall be evaluated and converted to the appropriate type for the conversion as specified below.

(9) The *format* operand shall be reused as often as necessary to satisfy the argument operands. Any extra `c` or `s` conversion specifications shall be evaluated as if a null string argument were supplied; other extra conversion specifications shall be evaluated as if a zero argument were supplied. If the *format* operand contains no conversion specifications and *argument* operands are present, the results are unspecified.

(10) If a character sequence in the *format* operand begins with a `%` character, but does not form a valid conversion specification, the behavior is unspecified.

The *argument* operands shall be treated as strings if the corresponding conversion character is `b`, `c`, or `s`; otherwise, it shall be evaluated as a C constant, as described by the C Standard {7}, with the following extensions:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— A leading plus or minus sign shall be allowed.

— If the leading character is a single- or double-quote, the value shall be the numeric value in the underlying code set of the character following the single- or double-quote.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message shall be written to standard error and the utility shall not exit with a zero exit status, but shall continue processing any remaining operands and shall write the value accumulated at the time the error was detected to standard output.

4.50.8 Exit Status

The `printf` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

4.50.9 Consequences of Errors

Default.

4.50.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage right, and print them out. The numbers are right-justified and separated by a single `<tab>`. The percentage is written to one decimal place of accuracy.

```
while read right wrong ; do
   percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
   printf "%2d right\t%2d wrong\t(%s%%)\n" \
           $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n" 1 21 321 4321 54321
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

produces:

```
     1  21
   3214321
 54321   0
```

Note that the *format* operand is used three times to print all of the given strings and that a `0` was supplied by `printf` to satisfy the last `%4d` conversion specification.

The `printf` utility is required to notify the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when `%d` is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
`5a`	`5`	`printf: "5a" not completely converted`
`9999999999`	`2147483647`	`printf: "9999999999" arithmetic overflow`
`-9999999999`	`-2147483648`	`printf: "-9999999999" arithmetic overflow`
`ABC`	`0`	`printf: "ABC" expected numeric value`

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the C Standard {7} function *strtol* (). A similar correspondence exists between `%u` and *strtoul* () and `%e`, `%f`, and `%g` (if the implementation supports floating-point conversions) and *strtod* ().

In a locale using ISO/IEC 646 {1} as the underlying code set, the command:

```
printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

produces:

`3`	Numeric value of constant 3
`3`	Numeric value of constant 3
`−3`	Numeric value of constant −3
`51`	Numeric value of the character ‘‘3’’ in ISO/IEC 646 {1} code set
`43`	Numeric value of the character ‘‘+’’ in ISO/IEC 646 {1} code set
`45`	Numeric value of the character ‘‘−’’ in ISO/IEC 646 {1} code set

Note that in a locale with multibyte characters, the value of a character is intended to be the value of the equivalent of the *wchar_t* representation of the character as described in C Standard {7}.

History of Decisions Made

The `printf` utility was added to provide functionality that has historically been provided by `echo`. However, due to irreconcilable differences in the various versions of `echo` extant, the version in this standard has few special features, leaving those to this new `printf` utility, which is based on one in the Ninth Edition at AT&T Bell Labs.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The Extended Description almost exactly matches the C Standard {7} *printf* () function, although it is described in terms of the file format notation in 2.12.

The floating point formatting conversion specifications are not required because all arithmetic in the shell is integer arithmetic. The `awk` utility performs floating point calculations and provides its own `printf` function. The `bc` utility can perform arbitrary-precision floating point arithmetic, but doesn't provide extensive formatting capabilities. (This `printf` utility cannot really be used to format `bc` output; it does not support arbitrary precision.) Implementations are encouraged to support the floating point conversions as an extension.

Note that this `printf` utility, like the C Standard {7}*printf* () function on which it is based, makes no special provision for dealing with multibyte characters when using the `%c` conversion specification or when a precision is specified in a `%b` or `%s` conversion specification. Applications should be extremely cautious using either of these features when there are multibyte characters in the character set.

Field widths and precisions cannot be specified as `'*'` since the `'*'` can be replaced directly in the *format* operand using shell variable substitution. Implementations can also provide this feature as an extension if they so choose.

Hexadecimal character constants as defined in the C Standard {7} are not recognized in the *format* operand because there is no consistent way to detect the end of the constant. Octal character constants are limited to, at most, three octal digits, but hexadecimal character constants are only terminated by a nonhex-digit character. In the C Standard {7}, the ## concatenation operator can be used to terminate a constant and follow it with a hexadecimal character to be written. In the shell, concatenation occurs before the `printf` utility has a chance to parse the end of the hexadecimal constant.

The `%b` conversion specification is not part of the C Standard {7}; it has been added here as a portable way to process backslash-escapes expanded in string operands as provided by the System V version of the `echo` utility. See also the rationale for `echo` for ways to use `printf` as a replacement for all of the traditional versions of the `echo` utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the `printf` utility is required to report an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion are to be reported as errors. If written in C, the `printf` utility could use the *strtol* () function to parse optionally signed numeric arguments, *strtoul* () to parse unsigned numeric arguments, and *strtod* () to parse floating point arguments (if floating point conversions are supported). It is not considered an error if an argument operand is not completely used for a `c` or `s` conversion or if a ‘‘string’’ operand's first or second character is used to get the numeric value of a character.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.51 `pwd` — Return working directory name

4.51.1 Synopsis

```
pwd
```

4.51.2 Description

The `pwd` utility shall write an absolute pathname of the current working directory to standard output.

4.51.3 Options

None.

4.51.4 Operands

None.

4.51.5 External Influences

4.51.5.1 Standard Input

None.

4.51.5.2 Input Files

None.

4.51.5.3 Environment Variables

The following environment variables shall affect the execution of `pwd`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.